An Evaluation of Knowledge Base Systems for Large OWL Datasets

Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin

Computer Science & Engineering Department, Lehigh University, Bethlehem, PA18015, USA
{yug2, zhp2, heflin}@cse.lehigh.edu

Technical Report: LU-CSE-04-012

Abstract. In this paper, we present our work on evaluating knowledge base systems with respect to use in large OWL applications. To this end, we have developed the Lehigh University Benchmark (LUBM). The benchmark is intended to evaluate knowledge base systems with respect to extensional queries over a large dataset that commits to a single realistic ontology. LUBM features an OWL ontology modeling university domain, synthetic OWL data generation that can scale to an arbitrary size, fourteen test queries representing a variety of properties, and a set of performance metrics. We describe the components of the benchmark and some rationale for its design.

Based on the benchmark, we have conducted an evaluation of four knowledge base systems (KBS). To our knowledge, no experiment has been done with the scale of data used here. The smallest dataset used consists of 15 OWL files totaling 8MB, while the largest dataset consists of 999 files totaling 583MB. We evaluated two memory-based systems (OWLJessKB and memory-based Sesame) and two systems with persistent storage (database-based Sesame and DLDB-OWL). We show the results of the experiment and discuss the performance of each system. In particular, we have concluded that existing systems need to place a greater emphasis on scalability.

1 Introduction

Various knowledge base systems (KBS) have been developed for processing Semantic Web information. They vary in a number of important ways. Many KBSs are main memory-based while others use secondary storage to provide persistence. Another key difference is the degree of reasoning provided by the KBS. Many systems are incomplete with respect to OWL [10], but still useful because they scale better or respond to queries quickly.

In this paper, we consider the issue of how to choose an appropriate KBS for a large OWL application. Here, we consider a large application to be one that requires the processing of megabytes of data. Generally, there are two basic requirements for such systems. First, the enormous amount of data means that scalability and efficiency become crucial issues. Second, the system must provide sufficient reasoning capabilities to support the semantic requirements of the application. However, increased reasoning capability usually means an increase in query response time as well. An impor-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Evaluation of Knowledge Base Systems for Large OWL Datasets	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Lehigh University,Computer Science & Engineering Department,19 Memorial Drive West,Bethlehem,PA,18015**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**27**	

tant question is how well existing systems support these conflicting requirements. Furthermore, different applications may place emphasis on different requirements.

It is difficult to evaluate KBSs with respect to these requirements, particularly in terms of scalability. The main reason for this is that there are few Semantic Web datasets that are of large size and commit to semantically rich ontologies. The Lehigh University Benchmark is our effort in order to fill this gap. We have developed the benchmark to facilitate the evaluation of those KBSs in a standard and systematic way. The benchmark contains a simulated ontology for the university domain and supports generating extensional data in arbitrary sizes. It offers fourteen test queries over the data. It also provides a set of performance metrics and related facilities for the evaluation.

By making use of the benchmark, we have performed an evaluation of four KBSs for the Semantic Web from several different aspects. We have evaluated two memory-based systems (OWLJessKB and memory-based Sesame) and two systems with persistent storage (database-based Sesame and DLDB-OWL). We present our experiment, discuss the performance of each system, and show some interesting observations. Based on that, we highlight some issues with respect to the development and improvement of the same kind of systems, and suggest some potential ways in using and developing those systems.

The outline of the paper is as follows: Section 2 elaborates on the Lehigh University Benchmark. Section 3 describes the aforementioned experiment and discusses the results. Section 4 talks about related work. Section 5 concludes.

2 Lehigh University Benchmark for OWL

We have developed the Lehigh University Benchmark (LUBM) to evaluate the performance of Semantic Web KBSs with respect to extensional queries over a large dataset that commits to a single realistic ontology. Extensional queries are queries about the instance data of ontologies. Recognizing that on the Semantic Web, data will by far outnumber ontologies, we wanted to develop a benchmark for this. Therefore, we chose to generate large amount of data for a single ontology of moderate size. LUBM was originally developed for the evaluation of DAML+OIL [9] repositories [13]. As OWL became the W3C recommendation, we have extended the benchmark to provide support for OWL ontologies and datasets. We introduce the key components of the benchmark suite below.

2.1 Benchmark Ontology

The ontology used in the benchmark is called Univ-Bench. Univ-Bench describes universities and departments and the activities that occur at them. Its predecessor is the Univ1.0 ontology[1], which has been used to describe data about actual universities and departments. We chose this ontology expecting that its domain would be familiar to

[1] http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml

most of the benchmark users. The ontology currently defines 43 classes and 32 properties.

We have created the Univ-Bench ontology OWL version [2]. The ontology is in OWL Lite, the simplest sublanguage of OWL. We chose to restrict the ontology (and also the test data) to OWL Lite since there are known complete and sound algorithms for the logic underpinning the language and are already some efficient reasoning systems available for it, e.g., Racer [16] and FaCT++[3].

As with its DAML+OIL version[4], the ontology contains specific language features that are useful for the benchmark. For instance, originally the Univ1.0 ontology states that GraduateStudent is a subclass of Student. In creating the Univ-Bench ontology, we have replaced that definition with what is shown in Fig. 1 using restriction. As a result, the subclass relationship between both the classes GraduateStudent and Student must be inferred using OWL semantics.

```
<owl:Class rdf:ID="GraduateCourse">
    <rdfs:label>Graduate Level Courses</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Course" />
</owl:Class>

<owl:Class rdf:ID="GraduateStudent">
    <rdfs:label>graduate student</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Person" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#takesCourse" />
            <owl:someValuesFrom>
                <owl:Class rdf:about="#GraduateCourse" />
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Student">
    <rdfs:label>student</rdfs:label>
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person" />
        <owl:Restriction>
            <owl:onProperty rdf:resource="#takesCourse" />
            <owl:someValuesFrom>
                <owl:Class rdf:about="#Course" />
            </owl:someValuesFrom>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

Fig. 1. Definition of the classes GraduateStudent and Student

[2] http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl
[3] http://owl.man.ac.uk/factplusplus/
[4] http://www.lehigh.edu//~zhp2/univ-bench.daml

In addition to the language change, there are some other differences between the ontology's OWL version and DAML+OIL version. First, we have used more RDFS vocabulary in the OWL ontology, e.g., *rdfs:domain* and *rdfs:range* in replace of *daml:domain* and *daml:range* respectively. Secondly, we have made some domain constraint changes to allow emphasis on description logic subsumption. For example, we have removed the domain constraint (to the class Student) of the property takesCourse so that no individuals of GradudateStudent in the benchmark data can be inferred as an instance of Student without the inference of the subsumption relationship between both classes.

2.2 Data Generation and OWL Datasets

LUBM's test data are extensional data created over the Univ-Bench ontology. In LUBM, we have adopted a method of synthetic data generation. This serves multiple purposes. As with the Wisconsin benchmark [3, 4], a standard and widely used database benchmark, this allows us to control the selectivity and output size of each test query. However, there are some other specific considerations:

1) We would like the benchmark data to be of a range of sizes including considerably large ones. It is hard to find such data sources that are based on the same ontology.
2) We may need the presence of certain kinds of instances in the benchmark data. This allows us to design repeatable tests for as many representative query types as possible. These tests not only evaluate the storage mechanisms for Semantic Web data but also the techniques that exploit formal semantics. We may rely on instances of certain classes and/or properties to test against those techniques.

Data generation is carried out by the Univ-Bench artificial data generator (UBA), a tool we have developed for the benchmark. In the tool, we have implemented the support for OWL datasets. The generator features random and repeatable data generation. A university is the minimum unit of data generation and for each university, a set of OWL files describing its departments are generated. Instances of both classes and properties are randomly decided. To make the data as realistic as possible, some restrictions are applied based on common sense and domain investigation. Examples are "a minimum of 15 and a maximum of 25 departments in each university", "an undergraduate student/faculty ratio between 8 and 14 inclusively", "each graduate student takes at least 1 but at most 3 courses", and so on and so forth. A detailed profile of the data generated by the tool can be found on the benchmark's webpage.

The generator identifies universities by assigning them zero-based indexes, e.g., the first university is named University0, and so on. Data generated by the tool are exactly repeatable in respect of universities. This is possible because the tool allows the user to enter an initial seed for the random number generator that is used in the data generation process. Through the tool, we may specify how many and which universities to generate.

Finally, as with the Univ-Bench ontology, the OWL data created by the generator are also in the OWL Lite sublanguage. As a consequence, we have had to give every

individual ID appearing in the data a type/class and include in every document an ontology tag (the *owl:Ontology* element)[5].

2.3 Test Queries

LUBM currently offers fourteen test queries, one more than when it was originally developed. Readers are referred to Appendix 1 for a list of these queries. In choosing the queries, first of all, we wanted them to be realistic. Meanwhile, we have mainly taken into account following factors:

1) *Input size.* This is measured as the proportion of the class instances involved in the query to the total class instances in the benchmark data. Here we refer to not just class instances explicitly expressed but also those that are entailed by the knowledge base. We define the input size as large if the proportion is greater than 5%, and small otherwise.
2) *Selectivity.* This is measured as the estimated proportion of the class instances involved in the query that satisfy the query criteria. We regard the selectivity as high if the proportion is lower than 10%, and low otherwise. Whether the selectivity is high or low for a query may depend on the dataset used. For instance, the selectivity of Queries 8, 11 and 12 is low if the dataset contains only University0 while high if the dataset contains more than 10 universities.
3) *Complexity.* We use the number of classes and properties that are involved in the query as an indication of complexity. Since we do not assume any specific implementation of the repository, the real degree of complexity may vary by systems and schemata. For example, in a relational database the number may directly indicate the times of join, which is a significant operation, or may not depending on the schema design.
4) *Assumed hierarchy information.* This considers whether information of class hierarchy or property hierarchy is required to achieve the complete answer. (We define completeness in next subsection).
5) *Assumed logical inference.* This considers whether logical inference is required to achieve the completeness of the answer. OWL features used in the test queries include subsumption, i.e., inference of implicit subclass relationship, *TransitiveProperty*, *inverseOf*, and realization, i.e., inference of the most specific concepts that an individual is an instance of. One thing to note is that we are not benchmarking complex description logic reasoning. We are concerned with extensional queries. Some queries use simple description logic reasoning mainly to verify that this capability is present.

We have chosen test queries that cover a range of types in terms of the above criteria. At the same time, to the end of performance evaluation, we have emphasized queries with large input and high selectivity. If not otherwise noted, all the test queries are of this type. Some subtler factors have also been considered in designing the queries,

---

[5] In OWL, the notion of the term ontology differs from that in the traditional sense by also including instance data [31].

such as the depth and width of class hierarchies[6], and the way the classes and properties chain together in the query.

To express the benchmark queries, we use a language in which a query is written as a conjunction of atoms. The language syntactically resembles KIF [12] but has less expressivity. We did not select from existing query language for RDF/OWL such as RQL [23], RDQL [29] or TRIPLE [30] since none of them has proven dominant. The simple language we use provides us with minimal while sufficient expressivity (i.e., existentially quantified conjunction of first-order logic atoms) and could be easily translated into any of the RDF/OWL query languages.

2.4 Performance Metrics

In addition, LUBM consists of a set of performance metrics including load time, repository size, query response time, query completeness and soundness, and a combined metric for the query performance. Among these metrics: the first three are standard database benchmarking metrics - query response time was introduced in the Wisconsin benchmark, and load time and repository size have been commonly used in other database benchmarks, e.g., the OO1 benchmark [8]; query completeness and soundness are new metrics we developed for the benchmark. We address these metrics in turn below.

Load Time

In a LUBM dataset, every university contains 15 to 25 departments, each described by a separate OWL file. These files are loaded to the target system in an incremental fashion. We measure the load time as the stand alone elapsed time for storing the specified dataset to the system. This also counts the time spent in any processing of the ontology and source files, such as parsing and reasoning.

Repository Size

Repository size is the consequent size of the repository after loading the specified benchmark data into the system. We only measure the consequent database sizes for the database based systems. We do not measure the occupied memory sizes for the main memory-based systems because it is difficult to accurately calculate them. However, since we evaluate all systems on a platform with a fixed memory size, the largest dataset that can be handled by a system provides an indication of its memory efficiency.

Query Response Time

Query response time is measured based on the process used in database benchmarks. To account for caching, each query is executed for ten times consecutively and the average time is computed. Specifically, the benchmark measures the query response time as the following:

For each target repository:
    For each test query:

---

[6] We define a class hierarchy as deep if its depth is greater than 3, and as wide if its average branching factor is greater than 3.

Open the repository.
Execute the query on the repository consecutively for 10 times and compute the average response time. Each time:
Issue the query, obtain the pointer to the result set, traverse that set sequentially, and collect the elapsed time.
Close the repository

Query Completeness & Soundness

We also examine query completeness and soundness of each system. In logic, an inference procedure is complete if it can find a proof for any sentence that is entailed by the knowledge base. With respect to queries, we say a system is complete if it generates all answers that are entailed by the knowledge base, where each answer is a binding of the query variables that results in an entailed sentence. However, on the Semantic Web, partial answers will often be acceptable. So it is important not to measure completeness with such a coarse distinction. Instead, we measure the degree of completeness of each query answer as the percentage of the entailed answers that are returned by the system. Note that we request that the result set contains unique answers.

In addition, as we will show in next section, we have realized that query soundness is also worthy of examination. With similar argument to the above, we measure the degree of soundness of each query answer as the percentage of the answers returned by the system that are actually entailed.

Combined Metric (CM)

The target systems in an evaluation may differ a lot in their inference capability. We feel it is insufficient to evaluate the query response time and answer completeness and soundness in isolation. We need a metric to measure them in combination so as to better appreciate the overall performance of a system and the potential tradeoff between the query response time and inference capability. At the same time, we have realized that this is a challenging issue. We introduce here our attempt to address this issue.

First, we use an F-Measure [28, 25] like metric to compute the tradeoff between query completeness and soundness, since essentially they are analogous to recall and precision in Information Retrieval. In the formula below, C_q and S_q ($\in [0, 1]$) are the answer completeness and soundness for query q. β determines the relative weighting of S_q and C_q. As will be shown in next section, some system might fail to answer a query. In that case, we will use F_q of zero in the calculation.

$$F_q = \frac{(\beta^2 + 1) * C_q * S_q}{\beta^2 * C_q + S_q}$$

Then, we define a composite metric CM of query response time and answer completeness and soundness as the following, which is also inspired by F-Measure:

$$CM = \frac{1}{M} \sum_{q=1}^{M} \frac{(\alpha^2 + 1) * P_q * F_q}{\alpha^2 * P_q + F_q}$$

In the above, M is the total number of test queries; $P_q \in [0, 1]$ is a query performance metric defined as

$P_q = \max (1 - \frac{T_q}{N}, \varepsilon)$

T_q is the response time (ms) for query q and N is the total number of triples in the dataset concerned. To allow for comparison of the metric values across datasets of different sizes, we use the response time per triple (i.e., $\frac{T_q}{N}$) in the calculation. Also we use a timeout value to eliminate undue affect of those query response time that is extremely far away from others in the test results: if to a certain query q, a system's response time per triple is greater than 1- ε, where ε is a very small positive value, we will use ε for P_q instead. α has the same role as β in F_q.

Generally speaking, the CM metric will reward those systems that can answer queries faster, more completely and more soundly.

2.5 Benchmarking Architecture

Fig. 2 depicts the architecture of the benchmarking. LUBM prescribes an interface to be instantiated by each target system. Through the interface, the benchmark test module launches the loading process, requests operations on the repository (e.g. open and close), issues queries and obtains the results. Users inform the test module of the target systems and test queries by defining them in the KBS configuration file and query definition file respectively. It needs to be noted that queries are translated from the above mentioned KIF-like language into the query language supported by the system prior to being issued to the system. In this way, we want to eliminate affect of query translation to the query response time. The translated queries are fed to the tester through the query definition file. The tester just reads the lines of each query from the definition file and passes them to the system.

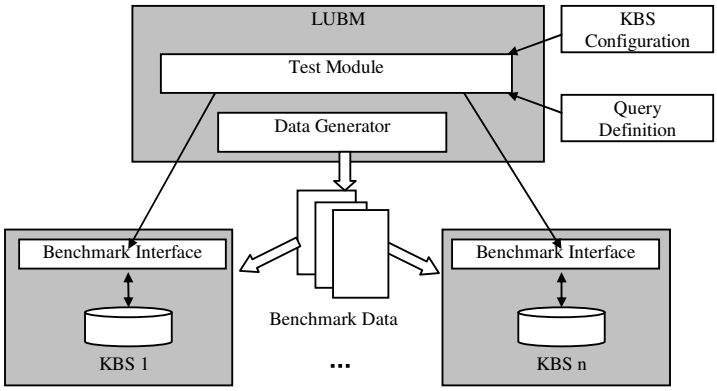

Fig. 2. Architecture of the benchmarking

The benchmark suite is accessible at http://www.lehigh.edu/~yug2/Research/SemanticWeb/LUBM/LUBM.htm.

3 An Evaluation Using LUBM

Using LUBM, we have conducted an evaluation of four systems. We describe the evaluation next.

3.1 Target Systems

In this experiment, we wanted to evaluate the scalability and support for OWL Lite in various systems. In choosing the systems, first we decided to consider only non-commercial systems. Moreover, we did not mean to carry out a comprehensive evaluation of the existing Semantic Web KBSs. Instead, we wanted to evaluate systems with a variety of characteristics. Additionally, we believe a practical KBS must be able to read OWL files, support incremental data loading, and provide programming APIs for loading data and issuing queries. As a result, we have settled on four different knowledge base systems, including two implementations of Sesame, OWLJessKB, and DLDB-OWL. We briefly describe each system below.

Sesame [6] is a repository and querying facility based on RDF and RDF Schema [34]. It features a generic architecture that separates the actual storage of RDF, functional modules offering operation on this RDF, and communication with these functional modules from outside the system. Sesame supports RDF/RDF Schema inference, but is an incomplete reasoner for OWL Lite. Nevertheless, it has been used on a wide number of Semantic Web projects. Sesame can evaluate queries in RQL, SeRQL[7], and RDQL. We evaluate two implementations of Sesame, main memory-based and database-based.

OWLJessKB, whose predecessor is DAMLJessKB [24], is a memory-based reasoning tool for description logic languages, particularly OWL. It uses the Java Expert System Shell (Jess) [21], a production system, as its underlying reasoner. Current functionality of OWLJessKB is close to OWL Lite plus some. We evaluate it as a system that supports most OWL entailments.

The fourth system, DLDB-OWL [27], is a repository for processing, storing, and querying large amounts of OWL data. Its major feature is the extension of a relational database system with description logic inference capabilities. Specifically, DLDB-OWL uses Microsoft Access® as the DBMS and FaCT [18] as the OWL reasoner. It uses the reasoner to precompute subsumption and employs relational views to answer extensional queries based on the implicit hierarchy that is inferred.

[7] http://www.openrdf.org/doc/users/ch06.html

Originally, we had targeted four other systems. The first is Jena [20], a Java framework for building Semantic Web applications. Jena currently supports both RDF/RDFS and OWL. We have done some preliminary tests on Jena (v2.1) (both memory-based and database-based) with our smallest dataset (cf. Appendix 3). Compared to Sesame, the most similar system to Jena here, Jena with RDFS reasoning was much slower in answering nearly all the queries. Some of the queries did not terminate even after being allowed to run for several hours. The situation was similar when Jena's OWL reasoning was turned on. For this reason, we have decided not to include Jena in the evaluation.

The second is KAON [22], an ontology management infrastructure. KAON provides an API for manipulating RDF models, however, it does not directly support OWL or RDFS in its framework.

We had also considered TRIPLE and Racer. TRIPLE [30] is an RDF query, inference, and transformation language and architecture. Instead of having a built-in semantics for RDF Schema, TRIPLE allows the semantics of languages on top of RDF to be defined with rules. For languages where this is not easily possible, TRIPLE also provides access to external programs like description logic classifiers. We were unable to test TRIPLE because it does not support incremental file loading and it does not provide a programming API either.

Racer [16] is a description logic inference engine currently supporting RDF, DAML+OIL and OWL. Running as a server, Racer provides inference services via HTTP or TCP protocol to client applications. Racer researchers have recently implemented a new query language nRQL. This language can be used to express all the current queries of LUBM and thus has made it possible to test Racer against the benchmark. In fact, they have already conducted such a performance evaluation of Racer (v1.8) using LUBM [17]. The results showed that Racer could offer complete answers for all the queries if required (they have tested Racer on Queries 1 through 13). However, since it has to perform Abox consistency check before query answering, Racer was unable to load a whole university dataset. As a result, they have only loaded up to 5 departments to Racer on a P4 2.8GHz 1G RAM machine running Linux. Due to this scalability limitation, we have decided not to re-test Racer.

Finally, we understand that there are other systems that could also fit into the benchmark. As noted above, this work is not intended to be a comprehensive evaluation of the existing KBSs. For some of those systems, we did not consider them because they are functionally close to one of the systems we have chosen, e.g., the ICS-FORTH RDFSuite [2] is similar to Sesame in that they are both an RDF store. We also understand that it is possible that these systems may outperform the systems we present here in some aspect. Those who are interested in evaluating these systems could always conduct an experiment with LUBM in a similar fashion.

3.2 Experiment Setup

System Setup

The systems we test are DLDB-OWL (04-03-29 release), Sesame v1.0, and OWLJessKB (04-02-23 release). As noted, we test both the main memory-based and data-

base-based implementations of Sesame. For brevity, we hereafter refer to them as Sesame-Memory and Sesame-DB respectively. For both of them, we use the implementation with RDFS inference capabilities. For the later, we use MySQL (v4.0.16) as the underlying DBMS since in a test by [6] Sesame performs significantly better than using the other DBMS PostgreSQL[8]. The DBMS used in DLDB-OWL is MS Access® 2002. We have created a wrapper over each system as an interface to the benchmark's test module.

Datasets

To identify the dataset, we use the following notation in the subsequent description:

LUBM(N, S): The dataset that contains N universities beginning at University0 and is generated using a seed value of S.

We have created 5 sets of test data[9]: LUBM(1, 0), LUBM(5, 0), LUBM(10, 0), LUBM(20, 0), and LUBM(50, 0), which contain OWL files for 1, 5, 10, 20, and 50 universities respectively, the largest one having over 6,800,000 triples in total. To our knowledge, prior to this experiment, Sesame has been tested with at most 3,000,000 statements. We have easily exceeded that by virtue of the benchmark supporting tool.

Query Test

For query test, the fourteen benchmark queries are expressed in RQL, Jess, and the KIF-like language and issued to Sesame, OWLJessKB, and DLDB-OWL respectively. As explained earlier, we do not use a common language in the test to eliminate affect of query translation to the query response time.

Query response time is collected in the way defined by the benchmark. Note that instead of providing a result set that can be iterated through, Sesame returns data one-at-a-time in streams and calls back user specified functions upon each result item. Thus we regard those call backs as the result traverse that is required by the benchmark, and count them in the query time instead.

As another detail, OWLJessKB only supports queries written in Jess language [21], and it needs two separate phrases to perform a query: define it and execute it. Interestingly, we have found out that the ordering of statements within a query can affect the response time of OWLJessKB to that query. It turned out that a direct translation of the benchmark queries resulted in poor performance from OWLJessKB after the one-university dataset is loaded. It even ran out of memory at some query (e.g., Query 2). However, if we do a reordering of the statements and put property related statements prior to the type related statements in each query, the response time could be reduced significantly. Therefore, in the experiment we use this non-standard approach to issue our test queries to OWLJessKB to get comparable results. Although we do it manually, such a reordering could be easily automated.

In addition, in our original experiment we have made use of the feature of Jess to pre-define the patterns for each test query prior to loading any data. However, we have newly found out that this could lead to worse performance of OWLJessKB. Since we

---

[8] http://www.postgresql.org

[9] The version of the data generator is UBA1.6.

have found no guidance as to when to or not to use such kinds of patterns, we will show the results of OWLJessKB with both settings in the subsequent discussion. When distinguishment is necessary, we will refer to them as OWLJessKB-P and OWLJessKB-NP respectively.

Test environment

We have done the test on a desktop computer. The environment is as follows:

1.80GHz Pentium 4 CPU;
256MB of RAM; 80GB of hard disk
Windows XP Professional OS;
Java SDK 1.4.1; 512MB of max heap size

In order to evaluate OWLJessKB, we needed to adjust this configuration slightly. With the standard setting for max heap size in Java, the system failed to load the one-university dataset due to out of memory errors. As a workaround, we increased the maximum heap size to 1GB, which requests large amount of virtual memory from operating system. This change allowed OWLJessKB to properly load the dataset.

3.3 Results and Discussions

3.3.1 Data Loading

Table 1. Load time and repository sizes

	Dataset	File #	Triple #	Load Time (hh:mm:ss)	Repository Size (KB)
DLDB-OWL	LUBM (1, 0)	15	103,397	00:05:43	16,318
Sesame-DB				00:09:02	48,333
Sesame-Memory				00:00:13	-
OWLJessKB-P				03:16:12	-
OWLJessKB-NP				02:19:18	-
DLDB-OWL	LUBM (5, 0)	93	646,128	00:51:57	91,292
Sesame-DB				03:00:11	283,967
Sesame-Memory				00:01:53	-
OWLJessKB				-	-
DLDB-OWL	LUBM (10, 0)	189	1,316,993	01:54:41	184,680
Sesame-DB				12:27:50	574,554
Sesame-Memory				00:05:40	-
OWLJessKB				-	-
DLDB-OWL	LUBM (20, 0)	402	2,782,419	04:22:53	388,202
Sesame-DB				46:35:53	1,209,827
Sesame-Memory				-	-
OWLJessKB				-	-
DLDB-OWL	LUBM (50, 0)	999	6,890,933	12:37:57	958,956
Sesame-DB				-	-
Sesame-Memory				-	-
OWLJessKB				-	-

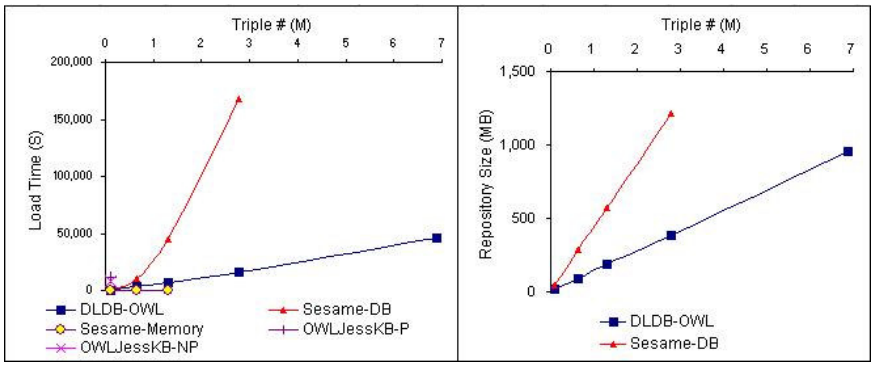

Fig. 3. Load time and repository sizes. The left hand figure shows the load time. The right hand figure shows the repository sizes of the database-based systems.

Table 1 shows the data loading time for all systems and the on-disk repository sizes of DLDB-OWL and Sesame-DB. Fig. 3 depicts how the load time grows as the data-set size increases and compares the repository sizes of the two database-based systems.

The test results have reinforced scalability as an important issue and challenge for Semantic Web knowledge base systems. One of the first issues is how large of a data-set each system can handle. As expected, the memory-based systems did not perform as well as the persistent storage systems in this regard. OWLJessKB, could only load the 1-university dataset, and took over 15 times longer than any other system to do so. On the other hand, we were surprised to see that Sesame-Memory could load up to 10 universities, and was able to do it in 5% of the time of the next fastest system. However, for 20 or more universities, Sesame-Memory also succumbed to memory limitations.

Using the benchmark, we have been able to test both Sesame-Memory and Sesame-DB on larger scale datasets than what has been reported so far. The result reveals an apparent problem for Sesame-DB: it does not scale in data loading, as can be seen from Fig. 3. As an example, it took over 300 times longer to load the 20-university dataset than the 1-university dataset, although the former set contains only about 25 times more triples than the later. We extrapolate that it will take Sesame-DB over 3 weeks to finish up loading the 50-university dataset. Therefore, we have decided not to do that unrealistic test.

In contrast, DLDB-OWL displays good scalability in data loading. We suspect the different performance of the two systems is caused by the following two reasons. First, to save space, both DLDB-OWL and Sesame map resources to unique IDs maintained in a table. When a resource is encountered during the data loading, they will look up that table to determine if it has not been seen before and needs to be assigned a new ID. As mentioned in [27], querying the ID table every time is very likely to slow down the data loading as the data size grows. In its implementation, Sesame also assigns every literal an ID, while DLDB-OWL stores literals directly in the destination tables, which means Sesame has to spend even more time on ID lookup. Moreover, in order

to improve performance, DLDB-OWL caches resource-ID pairs during current loading.

A second reason for the performance difference is related to the way Sesame performs inference. Sesame is a forward-chaining reasoner, and in order to support statement deletions it uses a truth maintenance system to track all deductive dependencies between statements. As [5] shows, this appears to affect the performance significantly if there are many inferred statements or the dataset is fairly large. We should note that this scalability problem was not as noticeable in our previous study involving a DAML+OIL benchmark [14]. We believe this is because the prior experiment used *daml:domain* (as opposed to *rdfs:domain*) in its ontology, which does not trigger inferences in Sesame.

3.3.2 Query Response Time

Readers are referred to Appendix 2 for a complete list of query test results including query response time, number of answers, and query completeness. Fig. 4 and Fig. 5 compares by graphs the query response time of the systems from two different views. Fig. 4 compares the performance of all the queries with respect to each dataset while Fig. 5 compares the query response time across all the datasets with respect to each query.

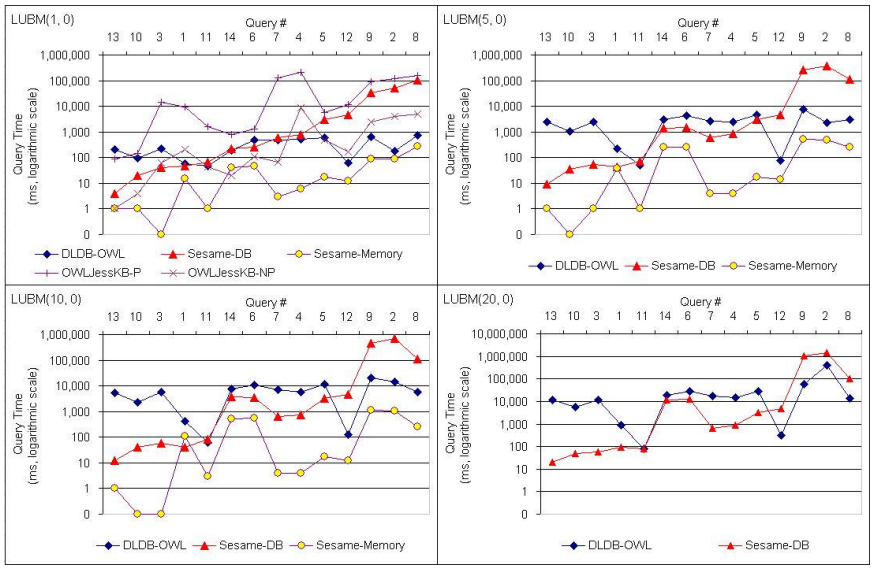

Fig. 4. Query response time comparison with respect to each dataset (up to 20 universities)

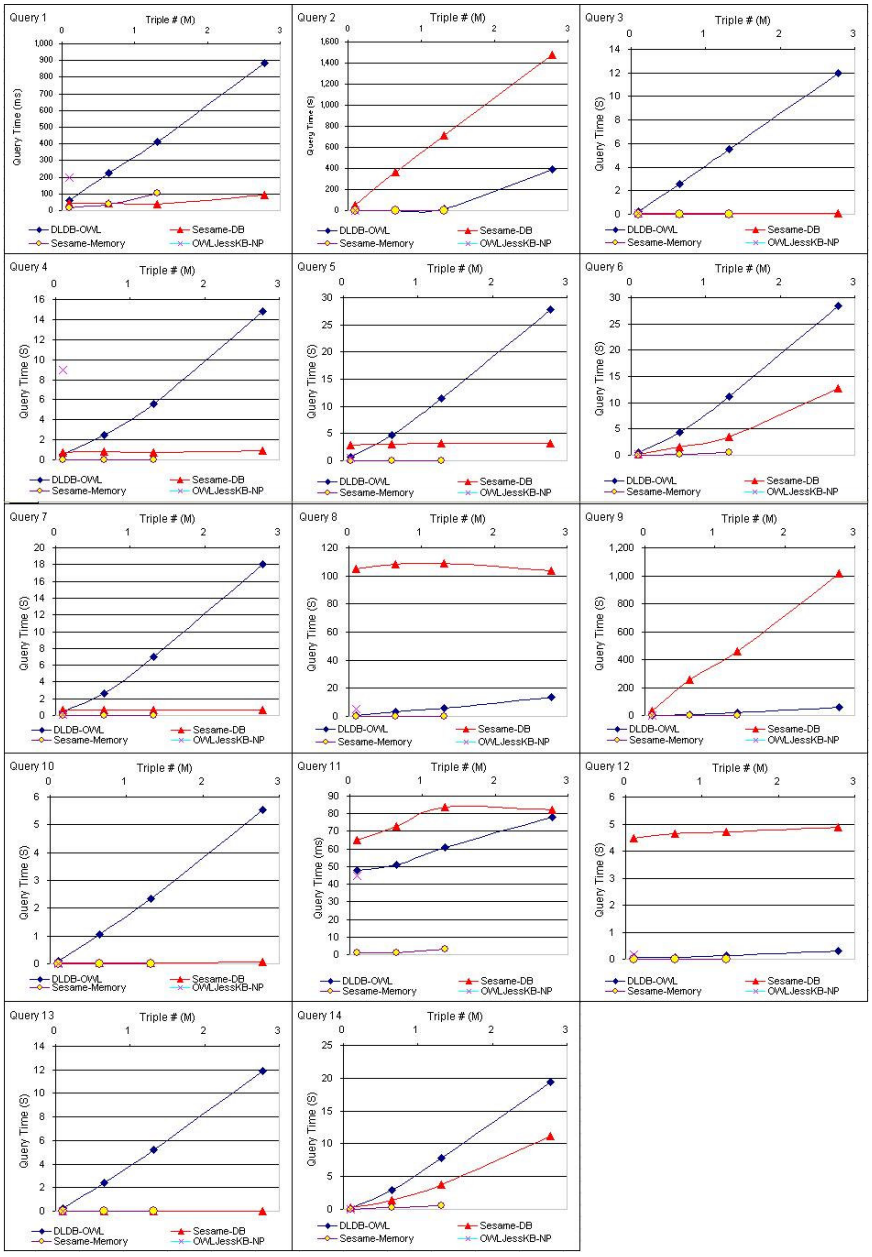

Fig. 5. Query response time comparison between DLDB-OWL, Sesame-DB, Sesame-Memory, and OWLJessKB-NP with respect to each query (up to 20 universities)

In terms of query, the results also lead to some scalability and efficiency concerns. Sesame-DB was very slow in answering some queries (even for one university), including Queries 2, 8, and 9. As for DLDB-OWL, it is the only system that has been tested with the largest dataset. One concern is that when it comes to the larger datasets especially the 50-university set, DLDB-OWL's query time no longer grows linearly for some queries, i.e., Queries 2, 5, 6, 7, 9, and 14. Moreover, it failed to answer Query 2 on the 50-univeristy dataset after MS Access ran out of temporary space. Regarding OWLJessKB, compared to the performance of its predecessor DAMLJessKB in [14], OWLJessKB improves its query time greatly at the sacrifice of much longer load time. Nonetheless, when OWLJessKB is queried with pre-defined patterns it is still the slowest in answering thirteen of the queries. However, it responds to the queries much faster when such patterns are not used and outperforms the database-based systems for quite a few queries. Compared to other systems, Sesame-Memory is the fastest in answering almost all the queries. It is also the fastest in data loading. This suggests that it might be the best choice for data of small scale if persistent storage and OWL inference is not required.

We have observed that those queries for which Sesame-DB's performance goes down dramatically are common in that they do not contain a specific URI as a subject or object in the statements. On the other hand, Sesame-DB shows a nice property in answering some other queries like Queries 3, 4, 5, 7, and 8: there was no proportional increase in the response time as the data size grows. We have also noticed a common feature of these queries, i.e., they have constant number of results over the test datasets. Whether these are the causes or coincidences is a subject for future work.

It is beyond the scope of this paper to analyze in depth the query evaluation and optimization mechanism in each system. Instead, we propose some topics for future investigation. One is to explore the potential relationship between query types and the performance of a certain system and its characteristics. Of course how to categorize queries is yet another issue. As another, Sesame-DB implements the main bulk of the evaluation in its RQL query engine while its query engine for another query language SeRQL pushes a lot of the work down to the underlying DBMS. As for DLDB-OWL, it directly translates as much of the query for the database. Further work should be done to investigate how these design differences as well as the underlying DBMS used impact performance.

3.3.3 Query Completeness and Soundness

It was noted before that we have chosen the benchmark test queries according to several criteria. In addition, we have made effort to make those queries as realistic as possible. In other words, we want these queries to represent, to some extent, those in the real world. We are very interested in seeing what queries can be answered by each system.

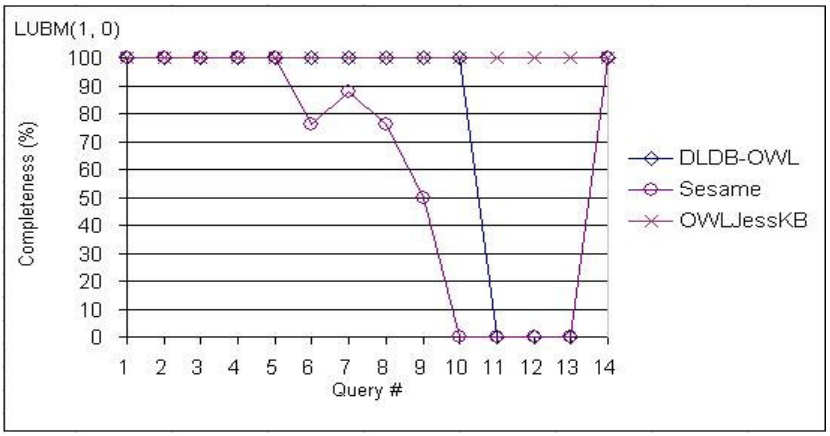

Fig. 6. Query completeness comparison. We show the results of only the first dataset since there are only minor if not no differences between the five datasets.

Fig. 6 depicts the comparison of query completeness between the systems. As mentioned, Sesame is able to address RDF/RDFS semantics while DLDB-OWL and OWLJessKB integrate extra OWL inference capability. As the results turned out, all systems could answer Queries 1 through 5 and Query 14 completely. As we expected, DLDB-OWL was able to find all the answers for Queries 6 to 10, which requires subsumption inference in order to get complete results, while Sesame could only find partial or no answers for them. It is interesting to notice that DLDB-OWL and Sesame found complete answers for Query 5 in different ways: DLDB-OWL made use of subsumption, while Sesame, although not able to figure out the subsumption, used an *rdfs:domain* restriction to determine the types of the individuals in the dataset and thus achieved the same result. OWLJessKB could find all the answers for every query, and was the only system to answer Queries 11 and 13 completely, which assume *owl:TransitiveProperty* and *owl:inverseOf* inference respectively. Nevertheless, we have discovered that OWLJessKB made unsound inferences with respect to some queries. Specifically, it returned incorrect answers to Queries 4, 6, 8, and 12 because it incorrectly inferred that Lecturer is a Professor, Employee a Student, and Student a Chair. We list in Table 2 the soundness of OWLJessKB for each query.

Table 2. Query soundness of OWLJessKB.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Correct answers /Total answers	4/4	0/0	6/6	34/41	719/719	7790/ 8330	67/67	7790/ 8330	208/20 8	4/4	224/22 4	15/540	1/1	5916/591 6
Soundness	100	100	100	83	100	94	100	94	100	100	100	3	100	100

3.3.4 Combined Metric Values

We have calculated the combined metric value of each target system with respect to each dataset. We use ε of 0.0001 in this evaluation. We set both β and α to 1, which means we equally weight query completeness and soundness, and also query response

time and F_q (cf. Section 2.4). Fig. 7 shows the results. We find that these numerical results are very helpful for us to appreciate the overall performance of each system. The higher values Sesame-Memory gets than Sesame-DB again suggest that it is a reasonable choice for small scale application if persistent storage is not required, particularly if completeness is not significant. DLDB-OWL achieves higher scores across all the datasets than Sesame. This helps us believe that its extra inference capability is not counterproductive. OWLJessKB-NP receives the highest value for the smallest dataset. However, the extremely long load time of OWLJessKB and its failure of loading larger datasets emphasize the need of performance improvement in that regard. Moreover, the great gap between the evaluation of OWLJessKB-P and OWLJessKB-NP suggests the necessity of a standard usage guidance of the system from the developers.

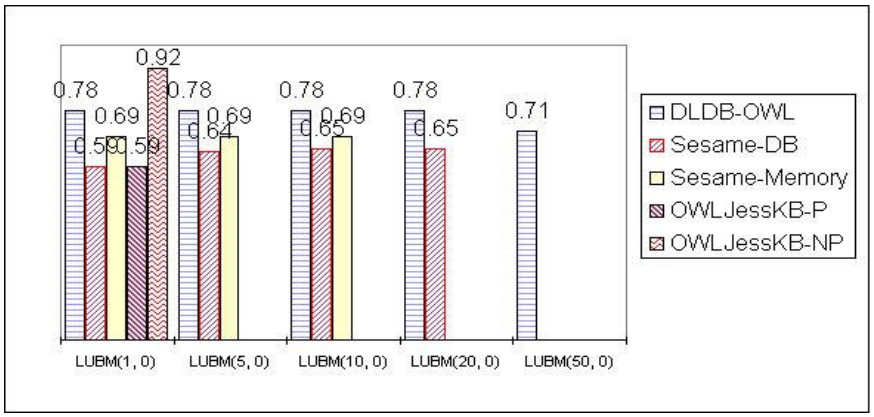

Fig. 7. CM values with weights β=1 and α=1

4 Related Work

To the best of our knowledge, the Lehigh University Benchmark is the first one for Semantic Web knowledge base systems in the area. There is a research work in benchmarking RDF schemata, which performs statistical analysis about the size and morphology of RDF schemata [26]. However this work does not provide a benchmark for evaluating a repository. In [1], they have developed some benchmark queries for RDF, however, these are mostly intensional queries, while we are concerned with extensional queries for OWL.

We have referred to several database benchmarks, including the Wisconsin benchmark [3, 4], the OO1 benchmark [8], and the SEQUOIA 2000 benchmark [32]. They are all DBMS-oriented benchmarks and storage benchmarks (vs. visualization benchmarks). LUBM shares in spirit with them methodology and rationale in terms of the use of synthetic data, some criteria for choosing test queries, and three of the performance metrics. However, our benchmark is tailored to the evaluation of OWL knowledge base systems and thus has many unique features. Particularly as shown in the

previous sections, the benchmark contains an ontology, data sets, test queries and criteria that reflect special concepts, structures and concerns in the Semantic Web area such as classes and properties, logical completeness vs. system performance, etc. Moreover, our benchmark is intended to work with any OWL repositories, not just database systems.

Some attempts have been done to benchmark description logic systems [11, 19]. The emphasis of this work is to evaluate the reasoning algorithms in terms of the tradeoff between expressiveness and tractability in description logic. Our benchmark is not a description logic benchmark. We are more concerned about the issue of storing and querying large amount of data that are created for realistic Semantic Web systems. In [11] and [19], they test the systems with respect to knowledge bases composed of a Tbox and an Abox, which can essentially be viewed as the counterparts of the ontology and the data set in our benchmark respectively. In [19] they use both artificial and realistic Tboxes and use synthetic Aboxes. But the Aboxes in the test are of fixed sizes. In contrast, our benchmark data can scale to arbitrary size. The Abox is randomly generated in [11]. However, unlike our benchmark data, the Abox is not customizable and repeatable. They also generate the Tbox randomly while our benchmark is based on a realistic ontology.

The Web Ontology Working Group provides a set of OWL test cases [7]. They are intended to provide examples for, and clarification of, the normative definition of OWL and focus on the completeness and soundness with respect to individual features. Our benchmark complements these tests. While these tests determine the capability and correctness of a system, our benchmark evaluates the performance of the system from different perspectives such as data loading, extensional queries and scalability.

In [33], they have done some preliminary work towards a benchmark for Semantic Web reasoners. Though their benchmark is still under construction, they analyze the publicly available ontologies and report them to be clustered into three categories. According to the characteristics of each category, our Univ-Bench ontology happens to be a synthetic "description logic-style" ontology, which has a moderate number of classes but several restrictions and properties per class. Therefore we argue that our evaluation represents at least a considerable portion of the real word situations. The other two categories are terminological ontologies and database schema-like ontologies. We are currently working on extending our benchmark suite to those two categories.

5 Conclusions

We presented our work on evaluating knowledge base systems (KBS) with respect to use in large OWL applications. We have developed the so-called Lehigh University Benchmark (LUBM) to standardize and facilitate such kind of evaluation. In LUBM: the Univ-Bench ontology models the university domain in OWL language and offers necessary features for the evaluation purpose; the data generator creates synthetic OWL datasets over the ontology. The synthetic data generated are random and repeat-

able, and can scale to an arbitrary size; Fourteen test queries are chosen to represent a variety of properties, including input size, selectivity, complexity, assumed hierarchy information, assumed logical inference, amongst others; A set of performance metrics are provided, which include load time & repository size, query response time, query completeness and soundness, and a combined metric for evaluating query performance. LUBM is intended to be used to evaluate Semantic Web KBSs with respect to extensional queries over a large dataset that commits to a single realistic ontology.

Using LUBM, we successfully conducted an evaluation of four systems, including two memory–based systems (OWLJessKB and memory-based Sesame) and two systems with persistent storage (database-based Sesame and DLDB-OWL). We tested those systems with 5 sets of benchmark data. To our knowledge, no experiment has been done with the scale of data used here. The smallest data size used consists of 15 OWL files totaling 8MB, while the largest data size consists of 999 files totaling 583MB.

It is clear that a number of factors must be considered when evaluating a KBS. From our analysis, of the systems tested: DLDB is the best for large datasets where an equal emphasis is placed on query response time and completeness. Sesame-Memory is the best when the size is relatively small (e.g., 1 million triples) and only RDFS inference is required; while for a larger dataset (e.g., between 1 and 3 million triples), Sesame-DB may be a good alternative. OWLJessKB is the best for small datasets when OWL Lite reasoning is essential, but only after its unsoundness has been corrected.

It should be pointed out that we believe that the performance of any given system will vary depending on the structure of the ontology and data used to evaluate it. Thus LUBM does not provide the final say on what KBS to use for an application. However, we believe that is appropriate for a large class of applications. Furthermore, the basic methodology can be used to generate ontologies and datasets for other classes of applications.

Acknowledgements

Some of the material in this paper is based upon work supported by the Air Force Research Laboratory, Contract Number F30602-00-C-0188 and by the National Science Foundation (NSF) under Grant No. IIS-0346963. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or NSF.

References

1. S. Alexaki, G. Karvounarakis, V. Christophides, D. Plexousakis, and K. Tolle. On Storing Voluminous RDF Description: The case of Web Portal Catalogs. In Proc. of the 4th International Workshop on the Web and Databases, 2001.

2. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The RDFSuite: Managing Voluminous RDF Description Bases. In Proc. of the 2nd International Workshop on the Semantic Web (SemWeb'01), in conjunction with the Tenth International World Wide Web Conference (WWW10), 2001.
3. D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking Database Systems, a Systematic Approach. In Proc. of the 9th International Conference on Very Large Data Bases, 1983
4. D. Bitton. and C. Turbyfill. A Retrospective on the Wisconsin Benchmark. In Readings in Database Systems, Second Edition, 1994.
5. J. Broekstra and A. Kampman. Inferencing and Truth Maintenance in RDF Schema: exploring a naive practical approach. In Workshop on Practical and Scalable Semantic Systems (PSSS), 2003.
6. J. Broekstra and A. Kampman. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proc. of the 1st International Semantic Web Conference (ISWC2002), 2002.
7. J.J. Carroll and J.D. Roo ed. OWL Web Ontology Test Cases, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-owl-test-20040210/
8. R.G.G. Cattell. An Engineering Database Benchmark. In Readings in Database Systems, Second Edition, 1994.
9. D. Connolly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein. DAML+OIL (March 2001) Reference Description. http://www.w3.org/TR/daml+oil-reference
10. M. Dean and G. Schreiber ed. OWL Web Ontology Language Reference, W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/
11. Q. Elhaik, M-C Rousset, and B. Ycart. Generating Random Benchmarks for Description Logics. In Proc. of DL'98, 1998.
12. M. Gensereth and R. Fikes. Knowledge Interchange Format. Stanford Logic Report Logic-92-1, Stanford Univ. http://logic.standford.edu/kif/kif.html
13. Y. Guo, J. Heflin, and Z. Pan. Benchmarking DAML+OIL Repositories. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), 2003.
14. Y. Guo, Z. Pan, and J. Heflin. Choosing the Best Knowledge Base System for Large Semantic Web Applications. In Proc. of the 13th International World Wide Web Conference (WWW2004) - Alternate Track Papers & Posters, 2004.
15. Y. Guo, Z. Pan, and J. Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In Proc. of the 3rd International Semantic Web Conference (ISWC2004), 2004.
16. V. Haarslev and R. Möller. Racer: A Core Inference Engine for the Semantic Web. In Workshop on Evaluation on Ontology-based Tools, the 2nd International Semantic Web (ISWC2003), 2003.
17. V. Haarslev, R. Möller, and M. Wessel. Querying the Semantic Web with Racer + nRQL. In Proc. of the Workshop on Description Logics 2004 (ADL2004).
18. I. Horrocks. The FaCT System. In Automated Reasoning with Analytic Tableaux and Related Methods International Conference (Tableaux'98).
19. I. Horrocks and P. Patel-Schneider. DL Systems Comparison. In Proc. of DL'98, 1998.
20. Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net/
21. Jess: the Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/jess
22. KAON: The KArlsruhe ONtology and Semantic Web tool suite. http://kaon.semanticweb.org/
23. G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. RQL: A Declarative Query Language for RDF. In Proc. of the Eleventh International World Wide Web Conference (WWW'02), 2002.
24. J.B. Kopena and W.C. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), 2003.

25. D.D. Lewis. A sequential algorithm for training text classifiers: Corrigendum and additional data. SIGIR Forum, 29(2), 13-19, 1995.
26. A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF schemas for the Semantic Web. In Proc. of the 1st International Semantic Web Conference (ISWC2002), 2002.
27. Z. Pan and J. Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In Workshop on Practical and Scalable Semantic Systems, the 2nd International Semantic Web Conference (ISWC2003), 2003.
28. C. J. van Rijsbergen. Information Retireval. Butterworths, London, 1979.
29. A. Seaborne. RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
30. M. Sintek and S. Decker. TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web. In Proc. of the 2nd International Semantic Web Conference (ISWC2002), 2002.
31. M.K. Smith, C. Welty, and D.L. McGuinness ed. OWL Web Ontology Language Guide, W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/
32. M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUIOA 2000 Storage Benchmark. In Readings in Database Systems, Second Edition, 1994.
33. C. Tempich and R. Volz. Towards a benchmark for Semantic Web reasoners–an analysis of the DAML ontology library. In Workshop on Evaluation on Ontology-based Tools, the 2nd International Semantic Web Conference (ISWC2003), 2003.
34. W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/

Appendix 1: Test Queries

We herein describe each query in the aforementioned KIF-like language. Following that we describe the characteristics of the query.

Query1

```
(type GraduateStudent ?X)
(takesCourse ?X http://www.Department0.University0.edu/GraduateCourse0)
```

This query bears large input and high selectivity. It queries about just one class and one property and does not assume any hierarchy information or inference.

Query2

```
(type GraduateStudent ?X)
(type University ?Y)
(type Department ?Z)
(memberOf ?X ?Z)
(subOrganizationOf ?Z ?Y)
(undergraduateDegreeFrom ?X ?Y)
```

This query increases in complexity: 3 classes and 3 properties are involved. Additionally, there is a triangular pattern of relationships between the objects involved.

Query3

```
(type Publication ?X)
```

(publicationAuthor
?X http://www.Department0.University0.edu/AssistantProfessor0)

This query is similar to Query 1 but class Publication has a wide hierarchy.

Query4

(type Professor ?X)
(worksFor ?X http://www.Department0.University0.edu)
(name ?X ?Y1)
(emailAddress ?X ?Y2)
(telephone ?X ?Y3)

This query has small input and high selectivity. It assumes *subClassOf* relationship between Professor and its subclasses. Class Professor has a wide hierarchy. Another feature is that it queries about multiple properties of a single class.

Query5

(type Person ?X)
(memberOf ?X http://www.Department0.University0.edu)

This query assumes *subClassOf* relationship between Person and its subclasses and *subPropertyOf* relationship between memberOf and its subproperties. Moreover, class Person features a deep and wide hierarchy.

Query6

(type Student ?X)

This query queries about only one class. But it assumes both the explicit *subClassOf* relationship between UndergraduateStudent and Student and the implicit one between GraduateStudent and Student. In addition, it has large input and low selectivity.

Query7

(type Student ?X)
(type Course ?Y)
(teacherOf http://www.Department0.University0.edu/AssociateProfessor0 ?Y)
(takesCourse ?X ?Y)

This query is similar to Query 6 in terms of class Student but it increases in the number of classes and properties and its selectivity is high.

Query8

(type Student ?X)
(type Department ?Y)
(memberOf ?X ?Y)
(subOrganizationOf ?Y http://www.University0.edu)
(emailAddress ?X ?Z)

This query is further more complex than Query 7 by including one more property.

Query9

(type Student ?X)

(type Faculty ?Y)
(type Course ?Z)
(advisor ?X ?Y)
(takesCourse ?X ?Z)
(teacherOf ?Y ?Z)

Besides the aforementioned features of class Student and the wide hierarchy of class Faculty, like Query 2, this query is characterized by the most classes and properties in the query set and there is a triangular pattern of relationships.

Query10

(type Student ?X)
(takesCourse ?X http://www.Department0.University0.edu/GraduateCourse0)

This query differs from Query 6, 7, 8 and 9 in that it only requires the (implicit) *subClassOf* relationship between GraduateStudent and Student, i.e., *subClassOf* relationship between UndergraduateStudent and Student does not add to the results.

Query11

(type ResearchGroup ?X)
(subOrganizationOf ?X http://www.University0.edu)

Query 11, 12 and 13 are intended to verify the presence of certain OWL reasoning capabilities in the system. In this query, property subOrganizationOf is defined as transitive. Since in the benchmark data, instances of ResearchGroup are stated as a suborganization of a Department individual and the later suborganization of a University individual, inference about the subOrgnizationOf relationship between instances of ResearchGroup and University is required to answer this query. Additionally, its input is small.

Query12

(type Chair ?X) (type Department ?Y)
(worksFor ?X ?Y)
(subOrganizationOf ?Y http://www.University0.edu)

The benchmark data do not produce any instances of class Chair. Instead, each Department individual is linked to the chair professor of that department by property headOf. Hence this query requires realization, i.e., inference that that professor is an instance of class Chair because he or she is the head of a department. Input of this query is small as well.

Query13

(type Person ?X)
(hasAlumnus http://www.University0.edu ?X)

Property hasAlumnus is defined in the benchmark ontology as the inverse of property degreeFrom, which has three subproperties: undergraduateDegreeFrom, mastersDegreeFrom, and doctoralDegreeFrom. The benchmark data state a person as an alumnus of a university using one of these three subproperties instead of hasAlumnus. Therefore, this query assumes *subPropertyOf* relationships between degreeFrom and its subproperties, and also requires inference about *inverseOf*.

Query14

(type UndergraduateStudent ?X)

This query is the simplest in the test set. This query represents those with large input and low selectivity and does not assume any hierarchy information or inference.

Appendix 2: Query Test Results

Table 3. Query test results [10]

Query	Repository & Data Set / Metrics	LUBM(1,0)					LUBM(5,0)			LUBM(10,0)			LUBM(20,0)		LUBM (50,0)
		DLDB-OWL	Sesame-DB	Sesame-Memory	OWL JessKB-P	OWL JessKB-NP	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	DLDB-OWL
1	Time(ms)	59	46	15	9203	200	226	43	37	412	40	106	887	96	2211
	Answers	4	4	4	4		4	4	4	4	4	4	4	4	4
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
2	Time(ms)	181	51878	87	116297	3978	2320	368423	495	14556	711678	1068	392392	1474664	failed
	Answers	0	0	0	0		9	9	9	28	28	28	59	59	-
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	-
3	Time(ms)	218	40	0	13990	164	2545	53	1	5540	59	0	11956	56	36160
	Answers	6	6	6	6		6	6	6	6	6	6	6	6	6
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
4	Time(ms)	506	768	6	211514	8929	2498	823	4	5615	762	4	14856	881	10115
	Answers	34	34	34	34*		34	34	34	34	34	34	34	34	34
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
5	Time(ms)	617	2945	17	5929	475	4642	3039	17	11511	3214	17	27756	3150	135055
	Answers	719	719	719	719		719	719	719	719	719	719	719	719	719
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100
6	Time(ms)	481	253	48	1271	112	4365	1517	251	11158	3539	543	28448	12717	151904
	Answers	7790	5916	5916	7790*		48582	36682	36682	99566	75547	75547	210603	160120	519842
	Completeness	100	76	76	100		100	76	76	100	76	76	100	76	100
7	Time(ms)	478	603	3	128115	67	2639	606	4	7028	634	4	18073	657	121673
	Answers	67	59	59	67		67	59	59	67	59	59	67	59	67
	Completeness	100	88	88	100		100	88	88	100	88	88	100	88	100
8	Time(ms)	765	105026	273	164106	4953	3004	108384	262	5937	108851	264	13582	103779	39845
	Answers	7790	5916	5916	7790*		7790	5916	5916	7790	5916	5916	7790	5916	7790
	Completeness	100	76	76	100		100	76	76	100	76	76	100	76	100

[10] The numbers marked with * do not count any incorrect answers returned by the system (refer to Section 3.3.3)

Table 4 continued

Query	Repository & Data Set / Metrics	LUBM(1,0)					LUBM(5,0)			LUBM(10,0)			LUBM(20,0)		LUBM (50,0)
		DLDB-OWL	Sesame-DB	Sesame-Memory	OWL JessKB-P	OWL JessKB-NP	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	Sesame-Memory	DLDB-OWL	Sesame-DB	DLDB-OWL
9	Time(ms)	634	34034	89	87475	2525	7751	256770	534	19971	460267	1123	57046	1013951	323579
	Answers	208	103	103	208		1245	600	600	2540	1233	1233	5479	2637	13639
	Completeness	100	50	50	100		100	48	48	100	49	49	100	48	100
10	Time(ms)	98	20	1	141	4	1051	36	0	2339	40	0	5539	50	15831
	Answers	4	0	0	4		4	0	0	4	0	0	4	0	4
	Completeness	100	0	0	100		100	0	0	100	0	0	100	0	100
11	Time(ms)	48	65	1	1592	45	51	73	1	61	84	3	78	82	143
	Answers	0	0	0	224		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
12	Time(ms)	62	4484	12	11266	162	78	4659	14	123	4703	12	310	4886	745
	Answers	0	0	0	15*		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
13	Time(ms)	200	4	1	90	1	2389	9	1	5173	12	1	11906	21	34854
	Answers	0	0	0	1		0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	100		0	0	0	0	0	0	0	0	0
14	Time(ms)	187	218	42	811	20	2937	1398	257	7870	3831[11]	515	19424	11175	106764
	Answers	5916	5916	5916	5916		36682	36682	36682	75547	75547	75547	160120	160120	393730
	Completeness	100	100	100	100		100	100	100	100	100	100	100	100	100

Appendix 3: Initial Test Results of Jena

The tables below show the initial test results of Jena (v2.1). We have tested Jena both based on the main memory (Jena-Memory) and using a MySQL database backend (Jena-DB). The benchmark queries were expressed in RDQL. We have tested Jena only with the smallest dataset. Unsurprisingly, when its RDFS reasoning was turned on, Jena's performance was exactly the same as Sesame's in terms of query completeness and soundness. However, Jena was much slower in answering most of the queries than Sesame. For some of the queries, Jena did not terminate even after being allowed to run for several hours. In this experiment we have used a timeout of 2 hours.

When Jena was used with its OWL inferencing, it could answer even smaller number of queries within the time limit. We speculate that the poor performance of Jena is

[11] This is an adjusted value from the original experiment [15], in which the query time was much longer. This was due to it happened that the OS was performing virtual memory increasing at the time of the query. We have updated the result without the affect of that operation.

due to that its rule-based reasoners are less optimized especially for a semantically complex ontology like Univ-Bench.

In order to investigate its query completeness and soundness with respect to the test queries, we have tested Jena with OWL reasoning on a single department file. This has allowed Jena to answer more queries within a reasonable time and noticeably, Jena could answer all those queries (including Queries 11-13) completely and correctly.

Table 4. Load time of Jena

	Dataset	Load Time (hh:mm:ss)
Jena-Memory (RDFS reasoning)	LUBM (1, 0)	00:00:12
Jena-DB (RDFS reasoning)		00:30:45
Jena-Memory (OWL reasoning)		00:00:13
Jena-DB (OWL reasoning)		00:32:27

Table 5. Query response time of Jena.

Query	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Jena-Memory (RDFS)	160	timeout	215	51	585	215	272951	timeout	timeout	209	14	4	203	220
Jena-DB (RDFS)	5715	timeout	13110	2860	24356	479	timeout	timeout	timeout	11562	536	1048	11731	2095
Jena-Memory (OWL)	time-out	timeout	3929	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	251
Jena-DB (OWL)	time-out	timeout	52818	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	timeout	2289

Table 6. Query completeness and soundness of Jena.

	Query	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Jena (RDFS, one university)	Completeness	100	n/a	100	100	100	76	88	n/a	n/a	0	0	0	0	100
	Soundness	100		100	100	100	100	100			100	100	100	100	100
Jena (RDFS, one department)	Completeness	100	100	100	100	100	78	88	78	38	0	0	0	0	100
	Soundness	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Jena (OWL, one department)	Completeness	100	n/a	100	100	100	100	n/a	n/a	n/a	n/a	100	100	100	100
	Soundness	100		100	100	100	100					100	100	100	100

n/a: not applicable due to timeout